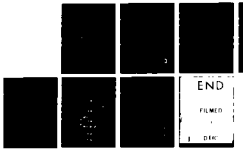
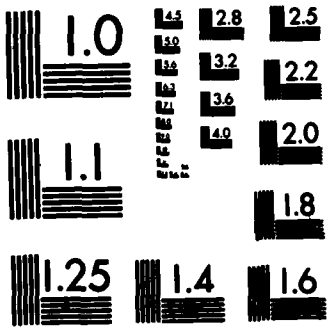
AD-A123 425 AN ACTIVATION/INHIBITION NETWORK CELL(U) ILLINOIS UNIV 1/1
AT URBANA COORDINATED SCIENCE LAB J POLLACK JAN 82
WP-31 N00014-75-C-0612

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A123425

AN ACTIVATION / INHIBITION NETWORK CELL

Jordan Pollack

Working Paper 31

Advanced Automation Research Group

Coordinated Science Laboratory

Urbana, Illinois

January, 1982

0. Abstract

Spreading Activation and Lateral Inhibition are highly parallel information processing techniques which have been used with some success in simulating human cognitive faculties on computers. Unfortunately, these simulations run like molasses on serial machines. This paper describes the design of a VLSI-based architecture for the parallel simulation of activation and inhibition.

This work was supported in part by the Office of Naval Research under contract N00014-75-C-0612.

DTIC FILE COPY

DTIC ELECTE JAN 17 1983 S D E

This document has been approved for public release and sale; its distribution is unlimited.

92 12 09 014

1. Introduction

Spreading Activation and Lateral Inhibition are highly parallel information processing techniques which have been used with some success in simulating human cognitive faculties on computers. Work on computer vision involving "relaxation" techniques[1], research in visual perception of letters and words[2], simulations of motor control[3], and research into the understanding of human memory[4] all use notions of activation and inhibition. The author's main research interest is in how these concepts are useful for Natural Language Processing.[5]

Activation and inhibition are highly parallel and local iterative processes which are applied to weighted networks. While they can be (and always have been) simulated on serial computers, where the compute time per iteration is linear with respect to the total number of links in the network, they also could easily be simulated on parallel machines, where the computation time per iteration would be constant.

We have completed most of the preliminary design and NMOS layout of a parallel machine for the simulation of an activation and inhibition network. Each node in a network maps into a single VLSI cell[6] which has storage for its activation level, its links to other nodes, and messages (i.e. contributions) to be sent. Each cell is physically connected to only two other cells, therefore, indirect messages are forwarded. The machine works in a cyclic fashion: Every cycle, each cell generates a bunch of messages and sends them all out, then updates its activation level based on the messages it received. A block diagram is shown in Figure 1.

---

[1] See, for example, [Waltz, 1975] or [Hinton,1977].

[2] [McClelland and Rumelhart, 1981] demonstrate how an activation/inhibition model can account for data from many studies of human perception of letters in context.

[3] [Rumelhart and Norman,1982] have an activation based model of a typist, complete with errors.

[4] Minsky's paper on K-lines [Minsky,1980] speculated that computation in the human brain may be organized as a society of interacting local agents, whose influence on each other occurs through activation and inhibition.

[5] See AARG Working Paper 35 for details.

[6] A current controversy in the field of parallel associative memories is whether there really should be a one to one mapping between nodes in an associative network and nodes in a physical network. Hinton [1981] gives some compelling arguments why there should not be. However, while all involved in the controversy agree that fully distributed memory is a powerful concept, no one has found a way of doing activation and inhibition on it, so I'll stick to the simple mapping.

1.1. BACKGROUND

The activation/inhibition network we are working with is a weighted and labeled directed graph, where the nodes represent concepts and the links represent binary relations between concepts. Node weights, $W_i(\tau)$, represent activation levels (a measure of relevance), and link weights, L_{ij}, represent strength of activation (if positive) or of inhibition (if negative). The processes of spreading activation and lateral inhibition involve the iterative recomputation of the activation level for each node based on its weighted connections. At each cycle τ, every node receives a contribution from each of its neighboring nodes equivalent to the neighbor's activation level multiplied by the weight of the intervening link:

$$C_i(\tau) = \sum_j W_j(\tau) \cdot L_{ij}$$

This contribution (scaled to range between -1 and 1) causes a proportional change in the activation level of the node for the next iteration:

$$W_i(\tau+1) = W_i(\tau) + \max(C_i(\tau),0) \cdot (M-W_i(\tau)) + \min(C_i(\tau),0) \cdot (W_i(\tau)-m)$$

So a contribution of 1 zaps the node up to its maximum activation level, M, while a contribution of -1 saps the node down to its minimum, m. Eventually, a static condition is reached where some nodes reach their maximum or minimum strength, while the rest of them receive contributions of 0, when the positive and negative contributions balance.

Serious use of activation/inhibition networks could require thousands of nodes and hundreds of such cycles. On a serial machine, execution will be very slow due to the multiple access of a very large memory; so slow, in fact, that cognitive researchers usually compromise their theories just to get acceptable run-times, thereby eliminating possibly important effects.

If an activation network were implemented in hardware, it would provide a vehicle for this type of research with real-time response, and could very well stimulate research in this area.

1.2. Problems for Design

There are many problems in hardware, however, which do not arise in software. The main costs in highly concurrent systems are connection and communication, not computation. Indeed, in a parallel simulation of an activation/inhibition network, every activation cycle will require a full barrage of messages to be sent, but relatively little computation.

The first problem is that of physical connectivity. It is well known that simple, regular interconnection schemes (i.e. tesselations of the plane, trees, etc.) are more feasible and less expensive than massively parallel (i.e. crossbar) interconnections[7]. Assuming

1 For
&I
ed
tion
ion/
Availability Codes
Dist | Avail and/or Special
A
DTIC COPY INSPECTED 2

a one node to one VLSI cell correspondence, there are many interconnections (i.e. links) which need to be made. Given that a simple and local physical interconnection scheme is used, the system must still support a large logical interconnection, i.e. processors must be able to send messages to non-adjacent processors. Logical interconnection must also deal effectively with message traffic to avoid bottlenecks and deadlock conditions.

A third interconnection problem is at the programming level; how to specify a subset of the logical interconnection to be used and how to reconfigure it.

Finally, the problem of central control is very important: Given a system which could exhibit a massive degree of parallelism, it would be inefficient (not to mention inelegant) to put it at the service of a central serial processor[8]. Related to the problem of central control is the problem of addressing: Are there any alternatives to fixed locations in a fixed address space?

1.3. A Satisficing[9] Design

The design for a activation/inhibition network cell described below provides satisfactory solutions to some of the aforementioned problems and sufficient solutions to others. There are 3 design parameters: α which is the address width, β which is the bit width for arithmetic, and μ which is the size of message queues. Using the simplest physical interconnectivity (linear adjacency) it can achieve a large logical interconnectivity (each processor can talk to $2^{\alpha+1}$ others) and get a full set of messages sent without possibility of saturation or deadlock in constant[10] time. Furthermore, the connections may be programmed quite easily, the addressing is fixed-width but relative to each cell, there is no central control, and the amount of real-estate is proportional to $\mu(\alpha+\beta)$.

This performance is achievable because the messages in an activation/inhibition network (i.e. neighborly contributions) are ultimately to be summed; so they may be summed as they are forwarded. If each node sends and receives two messages per cycle, and the incoming messages are merged with queued messages having the same destination, the number of messages queued cannot increase.

---

[7] Sutherland and Mead [1977] discuss the importance of having simple and regular geometries in VLSI.

[8] For example, Fahlman's [1979] design for a set-intersection machine is hampered by its connection to a central processor via a very overloaded bus.

[9] Satisfying + Sufficing = Satisficing, a term coined by Herbert Simon in his classic "The Sciences of the Artificial" [Cambridge: MIT Press, 1969].

[10] In this design, the time is $O(2^{\alpha})$. The best achievable message throughput would be $O(\mu)$, but this cannot be done with a simple, regular interconnection scheme.

Furthermore, if each node sends out its longest message first, then the length of the longest message is always decreasing. Since the maximum distance is $2^{\alpha}-1$, it is also the maximum number of cycles to complete a full activation cycle.

The system works with two two-phase, non-overlapping clocks. One of the clocks, ϕ, (on the order of the gate delay for an β-bit carry-propagate adder) is used for the internal clocking of the cells. The other, LOAD, has a short phase ($O(\mu\phi)$) used for loading the message queues and a long phase ($O(2^{\alpha})$) used for allowing the messages to completely propagate. Note that the problem of clock skew is minimal because of cell adjacency.

The basic message passing behavior of the cell is two- phase. In phase one of ϕ, messages are passed left while the next message going right is selected; in phase two, messages are passed to the right while the next message going left is selected. The message queues are dynamic sorting arrays in which alternating pairs of messages are sorted each alternating phase of ϕ.

An NMOS layout was performed for this cell in which the design parameters were α=4, β=4 and μ=8. The full layout is quite large (in order to see detail), so a condensed and outlined version is shown in Figure 2.

2. Description of Cell and Components

The cell is composed of two equal parts, one which handles messages going left to right, and the other handles messages going right to left. Each of these sub-processors is composed of three sections, the input/load section, the update section and the communication section.

The input/load section is responsible for loading the programmed connections (i.e. weighted links) into the communication section and accepting new links (from an unspecified external agent). While in the not-LOAD phase, new links may be presented and will be integrated into the (sorted) link memory. In the LOAD phase, the links are both passed into the update section to be multiplied by the current activation level and fed back around through μ delay stages into the sorted memory.

The update section is responsible for maintaining the activation level of the cell based on the current level and on the received messages which filter up from the communication section. This portion, which should be a microcoded arithmetic unit, has not been designed yet.

The communication section is responsible for message passing and merging. It contains a sorting memory which holds messages to go out, filters incoming messages to the top (which is input to the update section), and merges messages to be forwarded.

2.1. Input/Load Section

The main components of the input/load sections are the input/feedback selector, the delay stages, the sorting array,

and the load/hold selector. The input/feedback selector is a simple 1-of-2 selector based on the LOAD signal; if LOAD is true, it allows the (feedback) data from the bottom of the sorting array into the delay stages, if LOAD is false, it allows the input (new links) into the delay stages. The delay stages are simple dynamic registers which are clocked in both phases of ϕ. The load/hold selector controls the input to the multiplier and feedback; when LOAD is true, it feeds the bottom of the sort array back into itself and sends a 0 into the multiplier; when LOAD is false, it passes the output of the sort array into the multiplier (and feedback loop) and a 0 back into the sort array. The layouts for the two types of selector and the delay stage are shown on in Figure 3.

2.1.1. The Sort/Merge Array

The sort/merge array (or sorting memory) is the principle component in in the activation/inhibition network cell. Its goal is to maintain a list of address:weight pairs in sorted order, and it is used in several ways. In the input/load section of the cell, it is used as a merging memory for programmed links, and as a shift register which shifts out the longest (i.e. maximal relative address) link first. In the communication section, it is used as a message queue which filters up input messages, and sorts and merges output messages to minimize message delay and traffic.

It is essentially a very fancy up/down shift register which is composed bit-wise of α compare/swap cells and β add/swap cells sandwiched between dynamic registers. During alternate phases of the ϕ clock, each register is compared alternatively with the register above it and the one below it to see if they need merging or swapping. This is accomplished with two signals, ADD and SWAP, which are computed from the address portion of the registers.

Each of the four combinations of ADD and SWAP have meaning as follows, where A and B are the upper and lower registers being compared, and it is desired that messages with the same destination get added, but that incoming messages filter up.

ADD	SWAP	MEANING
0	0	(A < B)
0	1	(A > B)
1	0	(A = B) (not 0)
1	1	(A = B = 0)

These two signals are computed with NOR gates distributed through the compare/swap cell, and then are used to select outputs for the comparator cell and a corresponding adder cell in the following manner:

		COMPARATOR		ADDER	
ADD	SWAP	Aout	Bout	Aout	Bout
0	0	A	B	A	B
0	1	B	A	B	A
1	0	0	B	0	A+B
1	1	B	0	A+B	0

A demonstration of the use of the array as a sorter is given in the table below, where messages are represented as address:data and the pairs of messages being compared are boxed.

phase 1	1:2	5:4	3:0	0:1	1:3	2:4
phase 2	1:2	5:4	0:1	3:0	1:3	2:4
phase 1	1.2	0:1	5:4	1:3	3:0	2:4
phase 2	0:1	1:2	1:3	5:4	2:4	3:0
phase 1	0:1	0:0	1.5	2:4	5:4	3:0
phase 2	0:1	0:0	1:5	2:4	3:0	5:4

Note that besides the message with the greatest address (5:4) moving to the bottom (right side) of the array, the incoming message (0:1) moved to the top (left side), and the two messages with the same destination (1:2 and 1:3) merged into one (1:5).

The layouts for the compare/swap cell and the add/swap cell are shown in Figures 4 and 5.

2.2. The Update Section

The function of the update section is to collect the contributions arriving from the communications section and update the activation level register. Current work in activation/inhibition networks use thresholds to turn cells on and off, decay functions to avoid over-activation, and various other mechanisms. Before this cell could be fabricated, a real update section would have to be designed and built, most likely with a microprogrammed ALU or several operation specification bits.

2.3. The Communication Section

The last section of each half of the activation cell is the communications area. There is a sorting memory which holds, sorts and merges messages. Since each message contains a relative address, the address is decremented before the message is sent. Since the overlapped processes of message passing and merging never stops, empty (0:0) messages may get sent. For this reason, the decrementer needs to not decrement a zero, for that would cause a long, useless message to propagate.

The decrementer is bit slice design with a ripple carry and a distributed NOR gate for detecting the zero condition. The layout of a single bit decrementer cell is shown in Figure 6.

3. Further Work

More work needs to be done in order to see if this message merging process can be extended to a higher level of physical connectivity than two, and on the design and layout of a programmable activation level update function. Also, the load/input section needs a filter to remove links with zero weight.

4. Conclusion

Activation/inhibition networks are important techniques for research in cognitive simulation. While it is easy to simulate these networks on serial machines, serious studies of large networks can cause much thrashing. Given the dropping cost of building hardware and the need to find processing techniques which take advantage of the massive concurrency potential of SHSI[11], the union of these two technologies may be quite profitable. With that in mind, I have demonstrated how parallel simulation of activation/inhibition networks can be done, and done efficiently using a simple, regular structure in VLSI.

---

[11] SHSI = Super Humongus Scale Integration.

5. References

Fahlman, S.E. NETL: A system for representing and using real-world knowledge. MIT Press, Cambridge, 1979.

Hinton, G.E. Relaxation and its role in vision, unpublished doctoral dissertation, Edinburgh, 1977.

Hinton, G.E. Implementing Semantic Networks in Parallel Hardware in Parallel Models of Associative Memory Hinton, G.E. and J.A Anderson, Eds., Lawrence Erlbaum Associates, Hillsdale, N.J. 1981, pp 161-187.

McClelland, J.L and D.E. Rumelhart, An Interactive Activation Model of the Effect of Context in Perception Technical Reports 91 and 95, Center for Human Information Processing, UCSD, 1980

Minsky, M. K-lines: A theory of Memory Cognitive Science, 1980, V. 4, pp 117-133.

Rumelhart, J.L. and D.A. Norman, Simulating a Skilled Typist: A study of Skilled Cognitive-Motor Performance Cognitive Science, 1982, V. 6, pp. 1-36.

Simon, H.A., The Sciences of the Artificial. Cambridge: MIT Press, 1969.

Sutherland, I.E. and C.A. Mead, Microelectronics and Computer Science Scientific American, V. 237, N. 9, September, 1977, pp 210-228.

Waltz, D.L., Understanding Line Drawings of Scenes with Shadows in P. Winston, Ed. The Psychology of Computer Vision New York: McGraw Hill, 1975.

FIGURE 1

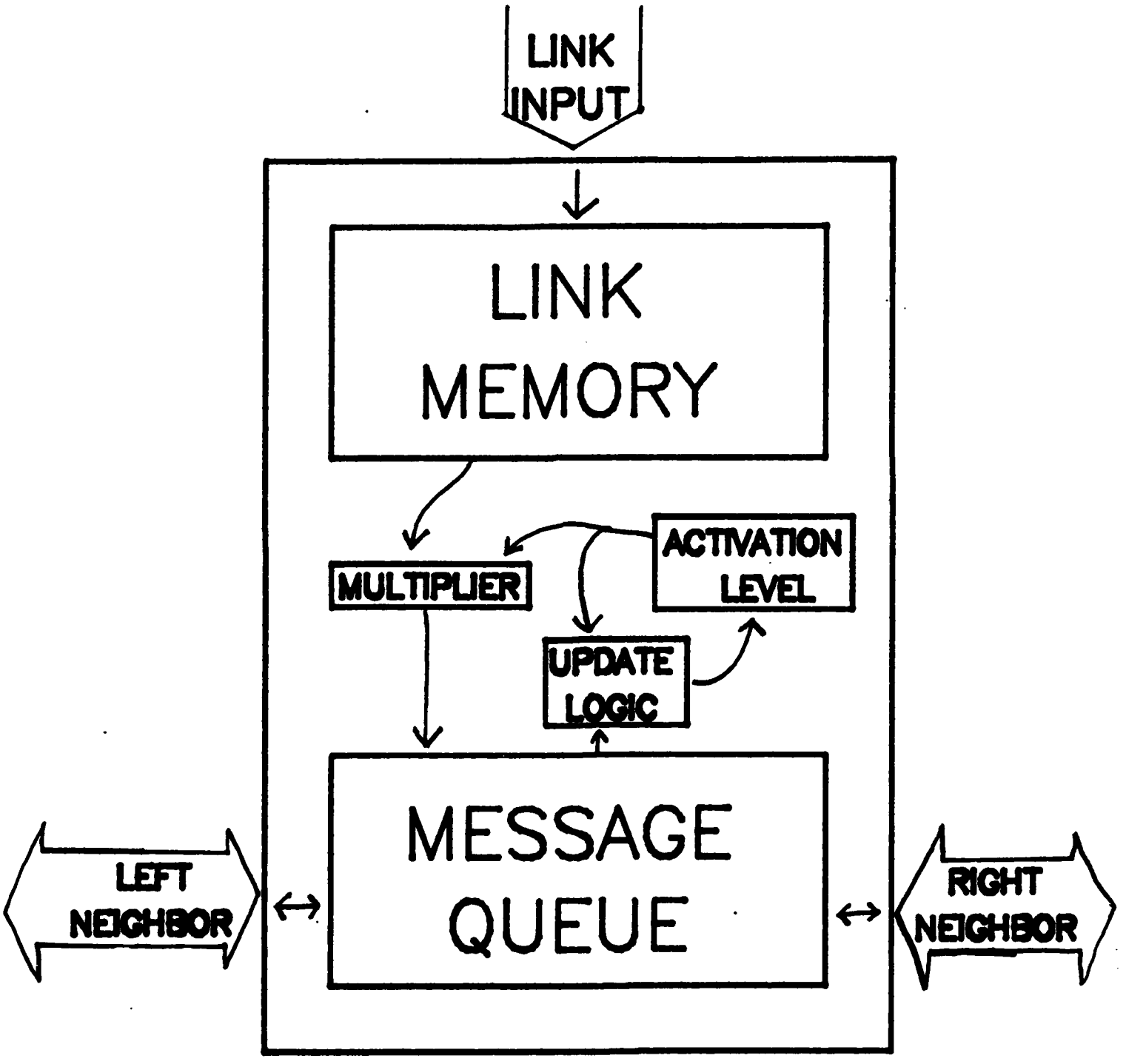

LINK
INPUT
LINK
MEMORY
MULTIPLIER
ACTIVATION
LEVEL
UPDATE
LOGIC
MESSAGE
QUEUE
LEFT
NEIGHBOR
RIGHT
NEIGHBOR

FIGURE 2

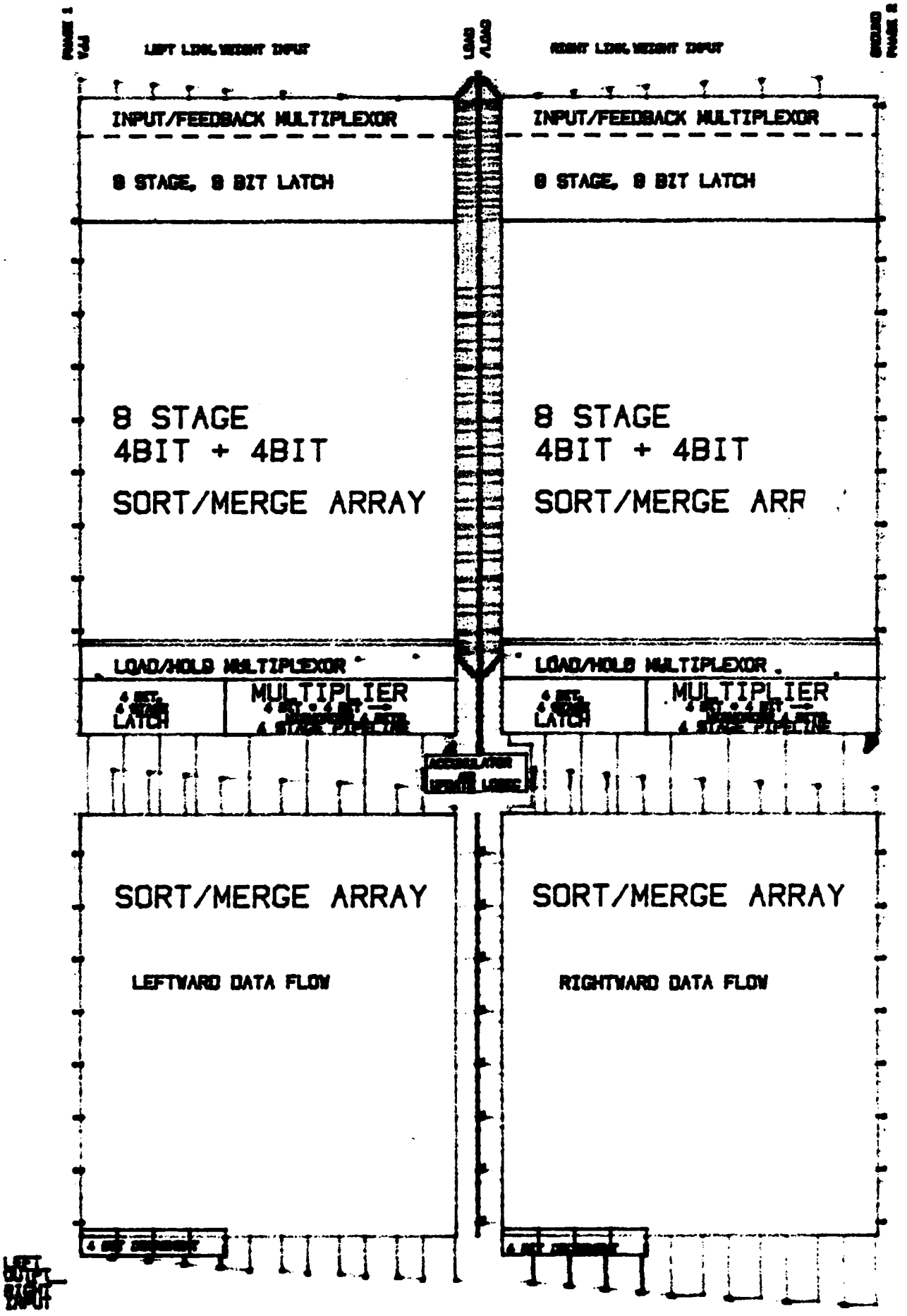

INPUT/FEEDBACK MULTIPLEXOR
8 STAGE, 8 BIT LATCH
INPUT/FEEDBACK MULTIPLEXOR
8 STAGE, 8 BIT LATCH
8 STAGE
4BIT + 4BIT
SORT/MERGE ARRAY
8 STAGE
4BIT + 4BIT
SORT/MERGE ARR
LOAD/HOLD MULTIPLEXOR
MULTIPLIER
LOAD/HOLD MULTIPLEXOR
MULTIPLIER
SORT/MERGE ARRAY
LEFTWARD DATA FLOW
SORT/MERGE ARRAY
RIGHTWARD DATA FLOW

FIGURE 3

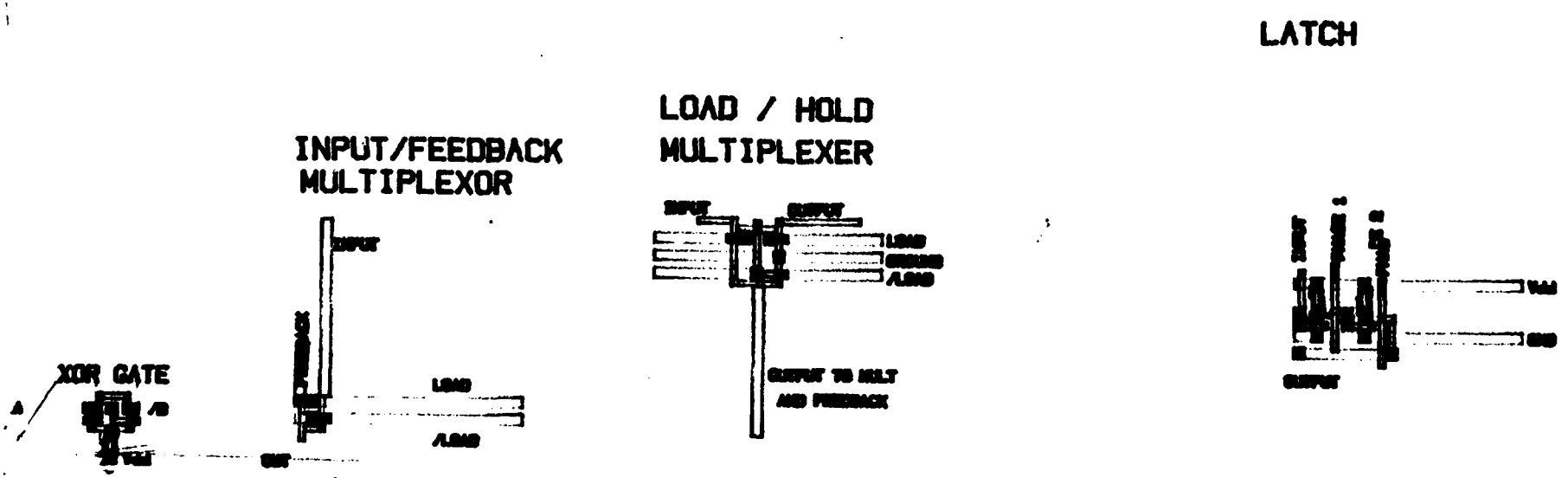

XOR GATE
INPUT/FEEDBACK
MULTIPLEXOR
LOAD / HOLD
MULTIPLEXER
LATCH

FIGURE 4

COMPARE/SWAP CELL

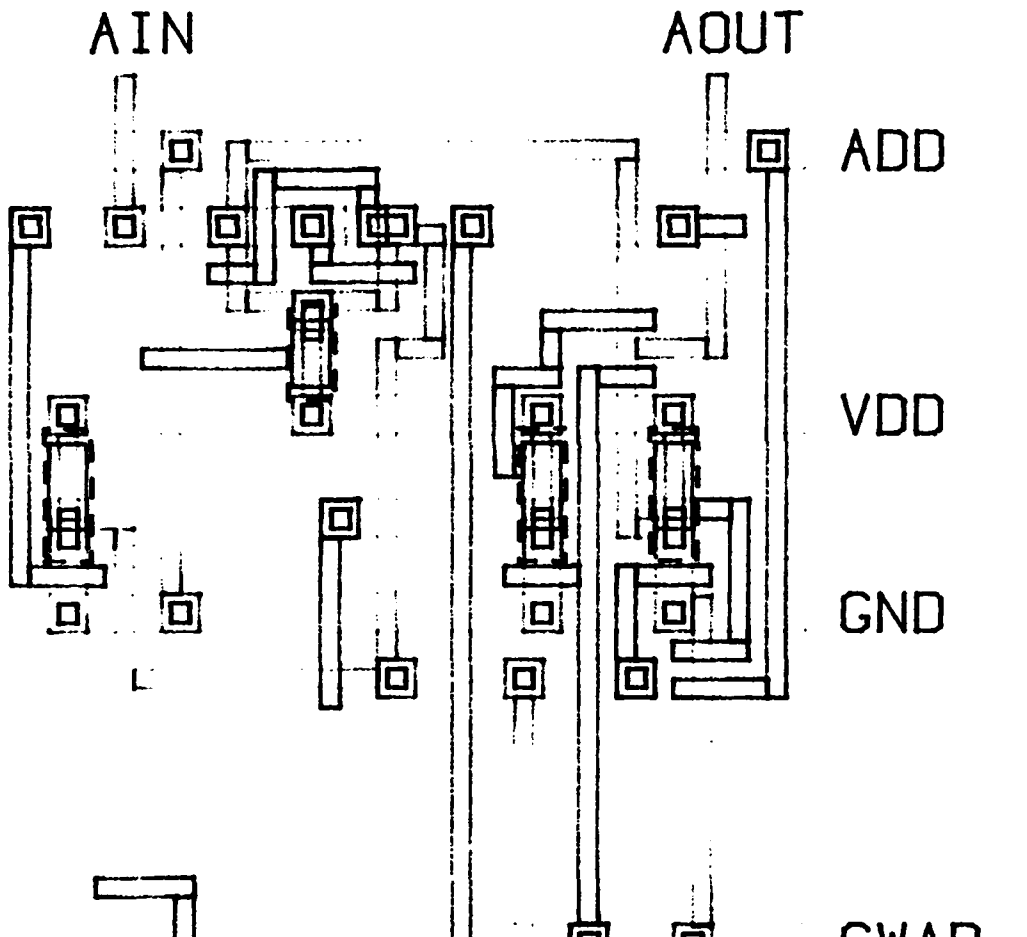


FIGURE 5

ADD/SWAP CELL

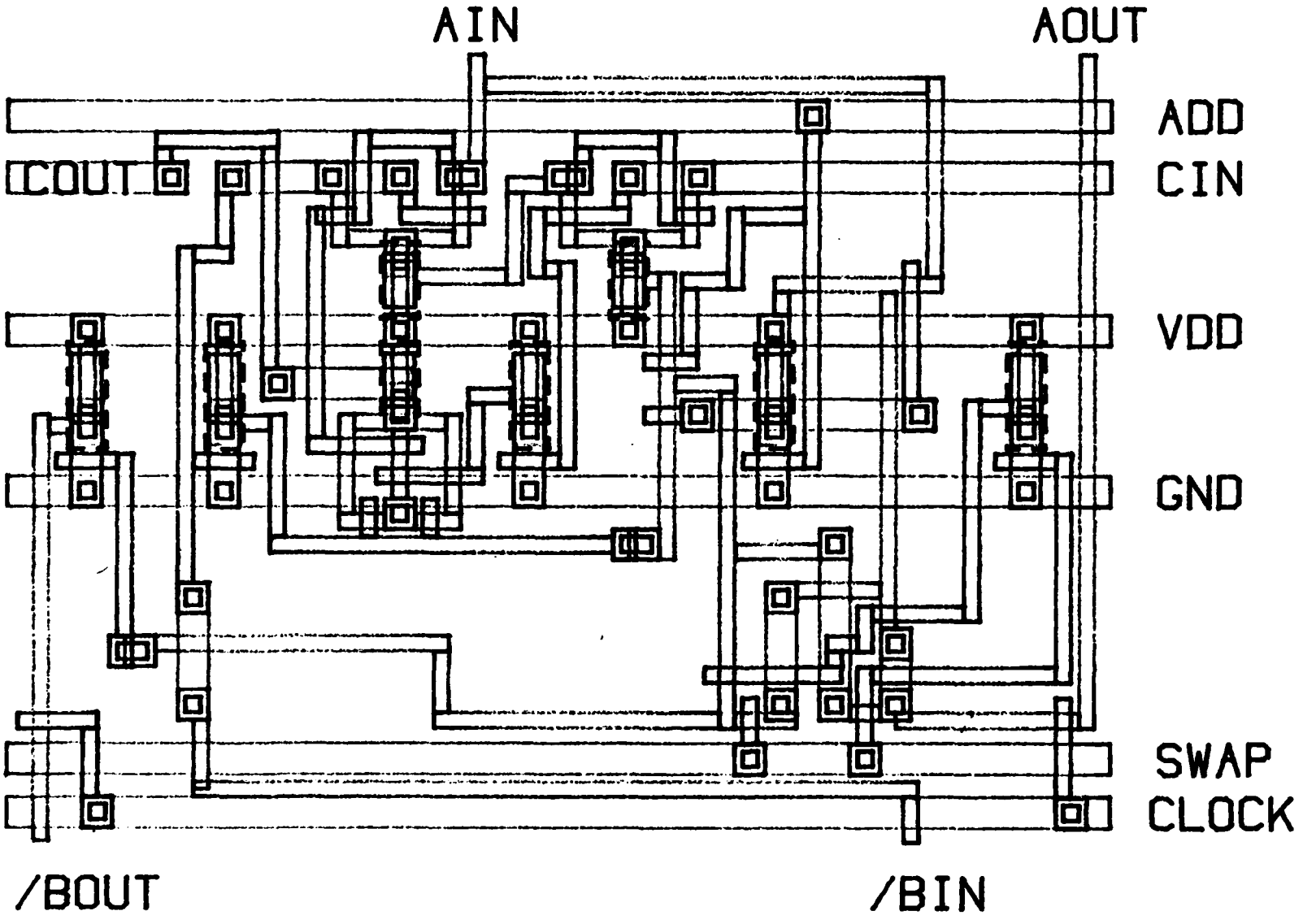
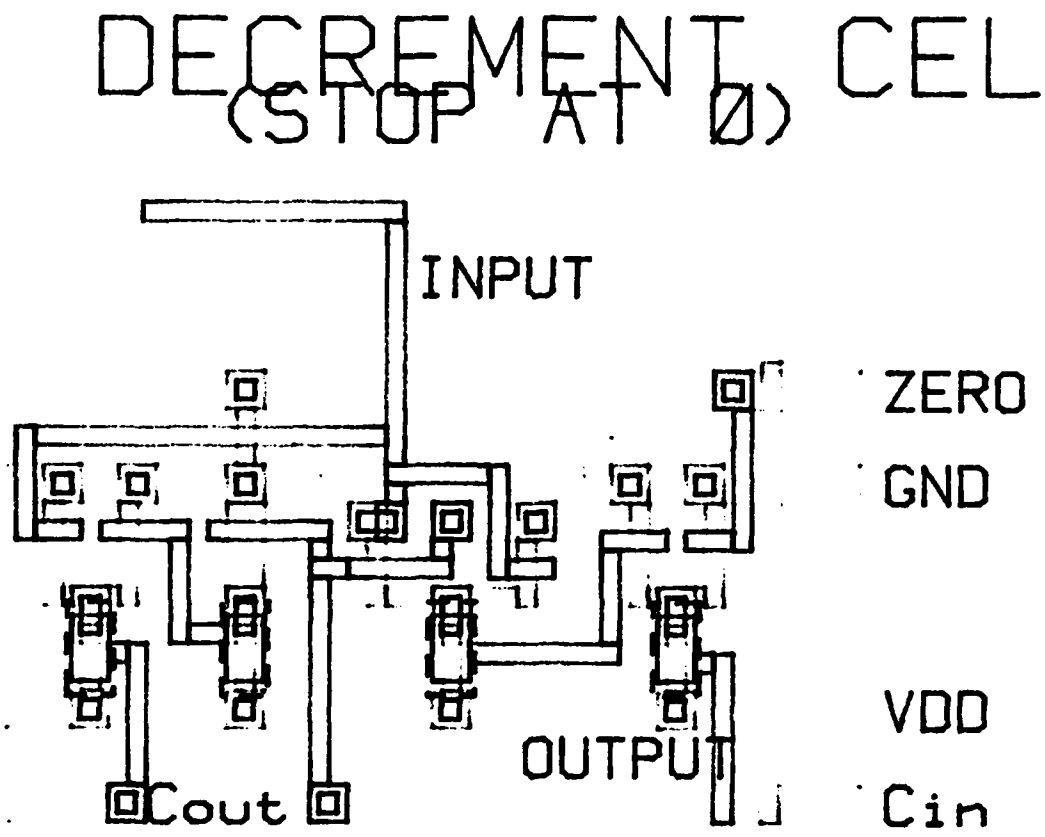
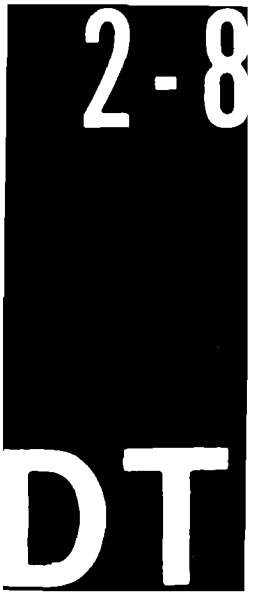


FIGURE 6




2-8
DT